When I Grow Up

Minimum experience: Grades K+, 1st year using ScratchJr, 2nd quarter or later

At a Glance

Overview and Purpose

Coders use a variety of blocks and sprites to create their own interactive diorama about when they grow up. The purpose of this project is to reinforce the concept of parallelism to trigger two or more algorithms simultaneously.

Objectives and Standards

Process objective(s):	Product objective(s):
Statement: • I will trigger two or more algorithms when a sprite is tapped. **Question:** • How can we trigger two or more algorithms when a sprite is tapped?	**Statement:** • I will create an interactive diorama that responds to a user. **Question:** • How can we create an interactive diorama that responds to a user?
Main standard(s):	**Reinforced standard(s):**
1A-AP-10 Develop programs with sequences and simple loops, to express ideas or address a problem. • Programming is used as a tool to create products that reflect a wide range of interests. Control structures specify the order in which instructions are executed within a program. Sequences are the order of instructions in a program. For example, if dialogue is not sequenced correctly when programming a simple animated story, the story will not make sense. If the commands to program a robot are not in the correct order, the robot will not complete the task desired. Loops allow for the repetition of a sequence of code multiple times. For example, in a program to show the life cycle of a butterfly, a loop could be combined with move commands to allow continual but controlled movement of the character. (source) **1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes. • Creating a plan for what a program will do clarifies the steps that will be needed to create a program and can be used to check if a program is correct. Students could create a planning document, such as a story map, a storyboard, or a sequential graphic organizer,	**1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks. • Composition is the combination of smaller tasks into more complex tasks. Students could create and follow algorithms for making simple foods, brushing their teeth, getting ready for school, participating in clean-up time. (source) **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. • Decomposition is the act of breaking down tasks into simpler tasks. Students could break down the steps needed to make a peanut butter and jelly sandwich, to brush their teeth, to draw a shape, to move a character across the screen, or to solve a level of a coding app. (source) **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. • Algorithms or programs may not always work correctly. Students should be able to use various strategies, such as changing the sequence of the steps, following the algorithm in a step-by-step manner, or trial and error to fix problems in algorithms and programs. (source) **1A-AP-15** Using correct terminology, describe steps taken and

to illustrate what their program will do. Students at this stage may complete the planning process with help from their teachers. (source)	choices made during the iterative process of program development. ● At this stage, students should be able to talk or write about the goals and expected outcomes of the programs they create and the choices that they made when creating programs. This could be done using coding journals, discussions with a teacher, class presentations, or blogs. (source)

Practices and Concepts

Source: K–12 Computer Science Framework. (2016). Retrieved from http://www.k12cs.org.

Main practice(s):	Reinforced practice(s):
Practice 5: Creating computational artifacts ● "The process of developing computational artifacts embraces both creative expression and the exploration of ideas to create prototypes and solve computational problems. Students create artifacts that are personally relevant or beneficial to their community and beyond. Computational artifacts can be created by combining and modifying existing artifacts or by developing new artifacts. Examples of computational artifacts include programs, simulations, visualizations, digital animations, robotic systems, and apps." (p. 80) ● **P5.1.** Plan the development of a computational artifact using an iterative process that includes reflection on and modification of the plan, taking into account key features, time and resource constraints, and user expectations. (p. 80) ● **P5.2.** Create a computational artifact for practical intent, personal expression, or to address a societal issue. (p. 80)	**Practice 6: Testing and refining computational artifacts** ● "Testing and refinement is the deliberate and iterative process of improving a computational artifact. This process includes debugging (identifying and fixing errors) and comparing actual outcomes to intended outcomes. Students also respond to the changing needs and expectations of end users and improve the performance, reliability, usability, and accessibility of artifacts." (p. 81) ● **P6.1.** Systematically test computational artifacts by considering all scenarios and using test cases." (p. 81) ● **P6.2.** Identify and fix errors using a systematic process. (p. 81) **Practice 7: Communicating about computing** ● "Communication involves personal expression and exchanging ideas with others. In computer science, students communicate with diverse audiences about the use and effects of computation and the appropriateness of computational choices. Students write clear comments, document their work, and communicate their ideas through multiple forms of media. Clear communication includes using precise language and carefully considering possible audiences." (p. 82) ● **P7.2.** Describe, justify, and document computational processes and solutions using appropriate terminology consistent with the intended audience and purpose. (p. 82)
Main concept(s):	**Reinforced concept(s):**
Program Development ● "Programs are developed through a design process that is often repeated until the programmer is satisfied with the solution. In early grades, students learn how and why people develop programs. As they progress, students learn about the tradeoffs in program design associated with complex decisions involving user constraints, efficiency, ethics, and testing." (p. 91)	**Algorithms** ● "Algorithms are designed to be carried out by both humans and computers. In early grades, students learn about age-appropriate algorithms from the real world. As they progress, students learn about the development, combination, and decomposition of algorithms, as well as the evaluation of competing algorithms." (p. 91)

• **Grade 2 -** "People develop programs collaboratively and for a purpose, such as expressing ideas or addressing problems." (p. 97)	• **Grade 2** - People follow and create processes as part of daily life. Many of these processes can be expressed as algorithms that computers can follow." (p. 96) **Control** • "Control structures specify the order in which instructions are executed within an algorithm or program. In early grades, students learn about sequential execution and simple control structures. As they progress, students expand their understanding to combinations of structures that support complex execution." (p. 91) • **Grade 2 -** "Computers follow precise sequences of instructions that automate tasks. Program execution can also be nonsequential by repeating patterns of instructions and using events to initiate instructions." (p. 96)

ScratchJr Blocks	
Primary blocks	Triggering
Supporting blocks	Control, End, Looks, Motion, Sound

Vocabulary	
Algorithm	• A step-by-step process to complete a task. (source) • A formula or set of steps for solving a particular problem. To be an algorithm, a set of rules must be unambiguous and have a clear stopping point. (source)
Debugging	• The process of finding and correcting errors (bugs) in programs. (source) • To find and remove errors (bugs) from a software program. Bugs occur in programs when a line of code or an instruction conflicts with other elements of the code. (source)
Event (trigger)	• An action or occurrence detected by a program. Events can be user actions, such as clicking a mouse button or pressing a key, or system occurrences, such as running out of memory. Most modern applications, particularly those that run in Macintosh and Windows environments, are said to be event-driven,because they are designed to respond to events. (source) • The computational concept of one thing causing another thing to happen. (source) • Any identifiable occurrence that has significance for system hardware or software. User-generated events include keystrokes and mouse clicks; system-generated events include program loading and errors. (source)
Loop	• A programming structure that repeats a sequence of instructions as long as a specific condition is true. (source) • In a loop structure, the program asks a question, and if the answer requires an action, it is performed and the original question is asked again until the answer is such that the action is no longer required. For example, a program written to compute a company's weekly payroll for each individual employee will begin by computing the wages of one employee and continue performing that action in a loop until there are no more employee wages to be computed, and only then will the program move on to its next action. Each pass through the loop is called an iteration. (source) • The computational concept of running the same sequence multiple times. (source)

Parallel	• Refers to processes that occur simultaneously. Printers and other devices are said to be either parallel or serial. Parallel means the device is capable of receiving more than one bit at a time (that is, it receives several bits in parallel). Most modern printers are parallel. (source) • The computational concept of making things happen at the same time. (source)
Parameter	• A specific piece of information used as input each time a process is run; for example, when a person presses a vending machine button for an item, the information for the item location is sent as a parameter so the machine knows what food to dispense. (Thank you, Peter Rich, for the great example) • A special kind of variable used in a procedure to refer to one of the pieces of data received as input by the procedure. (source) • In programming, the term parameter is synonymous with argument, a value that is passed to a routine. (source)
Sprite	• A media object that performs actions on the stage in a Scratch project. (source)
Storyboard	• Like comic strips for a program, storyboards tell a story of what a coding project will do and can be used to plan a project before coding.
More vocabulary words from CSTA	• Click here for more vocabulary words and definitions created by the Computer Science Teachers Association

Connections

Integration	**Potential subjects:** CTE, language arts, media arts **Example(s):** Rather than simply explaining what a coder might want to do when they grow up, you could modify the project to explore what a fictional character might want to do when they grow up. For example, creating a project about what Curious George might want to do when he grows up could connect with language arts standards.
Vocations	Media artists and designers are frequently asked to animate characters to create stories or animations. Click here to visit a website dedicated to exploring potential careers through coding.

Resources

- Project files
  - **Video:** Downloading project files (1:04)
- Sample project images

Project Sequence

Preparation (10+ minutes)

Suggested preparation	Resources for learning more

Ensure all devices are plugged in for charging over night.
 	
(10+ minutes) Read through each part of this lesson plan
	• BootUp ScratchJr Tips
	  ○ Videos and tips on ScratchJr from our YouTube channel
	• BootUp Facilitation Tips
	  ○ Videos and tips on facilitating coding classes from our YouTube channel
	• Block Descriptions
	  ○ A document that describes each of the blocks used in ScratchJr

and decide which sections the coders you work with might be interested in and capable of engaging with in the amount of time you have with them. If using projects with sound, individual headphones are very helpful.	• Interface Guide ◦ A reference guide that introduces the ScratchJr interface • Paint Editor Guide ◦ A reference guide that introduces features in the paint editor • Tips and Hints ◦ Learn even more tips and hints by the creators of the app • Coding as another language (CAL) ◦ A set of curriculum units for K-2 using both ScratchJr and KIBO robotics • ScratchJr in Scratch ◦ If you're using ScratchJr in Scratch, this playlist provides helpful tips and resources

Getting Started (5+ minutes)	
Suggested sequence	**Resources, suggestions, and connections**
1. Review and demonstration (2+ minutes): Begin by asking coders to talk with a neighbor for 30 seconds about something they learned last time; assess for general understanding of the practices and concepts from the previous project. Explain that today we are going to create an interactive diorama about what you want to do when you grow up. Display and demonstrate the sample project (or your own remixed version).	**Practices reinforced:** • Communicating about computing **Video:** Project Preview (1:03) **Video:** Lesson pacing (1:48) This can include a full class demonstration or guided exploration in small groups or individually. For small group and individual explorations, it might help to set a time limit for exploration before discussing the project. **Example review discussion questions:** • What's something new you learned last time you coded? ◦ Is there a new block or word you learned? • What's something you want to know more about? • What's something you could add or change to your previous project? • What's something that was easy/difficult about your previous project?
2. Discuss (3+ minutes): Have coders talk with each other about their favorite things (e.g., subjects, hobbies, activities, places, people, etc.). Next, discuss how they might create a project like the one demonstrated. If coders are unsure, and the discussion questions aren't helping, you can model thought processes: "I noticed the sprite moved around, so I think they used a motion block. What motion block(s) might be in the code? What else did you notice?" Another approach might be to wonder out loud by thinking aloud different algorithms and testing them out, next asking coders "what do you wonder about or want to try?" Review how to use two or more trigger blocks to trigger two or more algorithms at the same time. Remind everyone they can use this approach to make something talk and move at the same time.	**Practices reinforced:** • Communicating about computing **Note:** Discussions might include full class or small groups, or individual responses to discussion prompts. These discussions which ask coders to predict how a project might work, or think through how to create a project, are important aspects of learning to code. Not only does this process help coders think logically and creatively, but it does so without giving away the answer. **Example discussion questions:** • What would we need to know to make something like this in ScratchJr? • What kind of blocks might we use? • What else could you add or change in a project like this?

After the discussion, coders will begin working on their project as a class, in small groups, or at their own pace.	● What code from our previous projects might we use in a project like this?
	● What do you want to learn when you grow up?
	    ○ What kind of job do you want to have?
	    ○ Where do you want to go or live?
	    ○ How do you want to help others?
	● What kind of sprites might we see in in a project about when you grow up?
	    ○ What kind of code might they have?

Project Work (40+ minutes; 2+ classes)	
Suggested sequence	**Resources, suggestions, and connections**

<u>3. Create a storyboard (10+ minutes):</u>
Either hand out paper with at least four different quadrants (one for each page in ScratchJr), use handheld whiteboards, or use a painting app on a device to encourage coders to storyboard what they are going to create. It may help to model this process with a separate set of random ideas.
 	
Encourage coders to draw or write out not only the kinds of sprites and backgrounds they're going to use, but the kind of code that will accompany them.
 	
When coders are ready, have them show you their storyboard and ask questions for clarification of their intent (which may change once they start coding and get more ideas). If approved, they may continue on to the next step (creating); otherwise they can continue to think through and work on their storyboard.
 	
Note: Coders may change their mind midway through a project and wish to rethink through their original storyboard. This is part of the design process and it is encouraged they revise their storyboard to reflect their new ideas.
	Standards reinforced:
	● **1A-AP-12** Develop plans that describe a program's sequence of events, goals, and expected outcomes
	Practices reinforced:
	● Creating computational artifacts
	Concept reinforced:
	● Program development
	
	Resource: Example storyboard templates
	Suggested storyboard questions:
	● What do you want to learn when you grow up?
	    ○ What kind of job do you want to have?
	    ○ Where do you want to go or live?
	    ○ How do you want to help others?
	● What might we include in our storyboard for creating an interactive diorama?
	    ○ What are some of the media we might use in a project like this (pictures, sounds, sprites, backdrops, etc.)?
	● What are all of the ways we can interact with the diorama?
	    ○ In each of these ways we can interact with the diorama, how might we use code to create that interaction?
	
	Suggestion: If coders need additional help, perhaps pair them with someone who might help them with the storyboarding process. Or, you could have coders meet with a peer to discuss their storyboard before asking to share it with yourself. This can be a great way to get academic feedback and ideas from a peer.
	
	Note: This process may take significantly longer than 10 minutes if storyboarding many different sprites and pages (not recommended until the end of the first year).

<u>4. Create an interactive diorama about when you grow up (30+ minutes):</u> Ask coders to create a project based on the storyboards they created for their interactive diorama about when they grow	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Testing and refining computational artifacts

up. Facilitate by walking around and asking questions and encouraging coders to try out new blocks. When coders complete what they have already storyboarded, ask them to storyboard some more ideas and repeat the process from step 3 and step 4.	• Creating computational artifacts **Concepts reinforced:** • Algorithms • Control **Note:** Coders might use the play recorded sound blocks to record and play their dialogue, sounds, and music, or they might use the say block by typing in their text or using speech to text recognition technologies.

Assessment

Standards reinforced:

- **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development

Practices reinforced:

- Communicating about computing

Although opportunities for assessment in three different forms are embedded throughout each lesson, this page provides resources for assessing both processes and products. If you would like some example questions for assessing this project, see below:

Summative Assessment *of* Learning	**Formative** Assessment *for* Learning	**Ipsative** Assessment *as* Learning
The debugging exercises, commenting on code, and projects themselves can all be forms of summative assessment if a criteria is developed for each project or there are "correct" ways of solving, describing, or creating. **For example, ask the following after a project:** • Can coders debug the debugging exercises? • Did coders create a project similar to the project preview? ○ **Note:** The project preview and sample projects are not representative of what all grade levels should seek to emulate. They are meant to generate ideas, but expectations should be scaled to match the experience levels of the coders you are working with. • Did coders use a variety of block types in their algorithms and can they explain how they work together for specific purposes? • Can coders explain how their	The 1-on-1 facilitating during each project is a form of formative assessment because the primary role of the facilitator is to ask questions to guide understanding; storyboarding can be another form of formative assessment. **For example, ask the following while coders are working on a project:** • What are three different ways you could change that sprite's algorithm? • What happens if we change the order of these blocks? • What could you add or change to this code and what do you think would happen? • How might you use code like this in everyday life? • See the suggested questions throughout the lesson and the assessment examples for more questions.	The reflection and sharing section at the end of each lesson can be a form of ipsative assessment when coders are encouraged to reflect on both current and prior understandings of concepts and practices. **For example, ask the following after a project:** • How is this project similar or different from previous projects? • What new code or tools were you able to add to this project that you haven't used before? • How can you use what you learned today in future projects? • What questions do you have about coding that you could explore next time? • See the reflection questions at the end for more suggestions.

project is similar to their storyboard? ● Did coders trigger multiple algorithms with a single event type (trigger blocks)? ● Can coders explain how their algorithms match what they want to do when they grow up? ● Did coders create an interactive project about when they grow up with at least ## different sprites with different algorithms? ○ Choose a number appropriate for the coders you work with and the amount of time available.		

Extended Learning

Project Extensions	
Suggested extensions	**Resources, suggestions, and connections**
Reverse engineering our example project (10+ minutes each): ***1 minute intro demonstration*** Demonstrate one of the following example sprites on the board without displaying the code (they are in order of complexity): ● Sun ● Shop ● Ball ● Rocket ● Boat ● Pilot ● Child ● Whale ● Bird ***4+ minute reverse engineering and peer-to-peer coaching*** Ask coders to see if they can figure out how to use their code blocks to create an algorithm that makes a sprite do something similar to what was demonstrated. Facilitate by walking around and asking guiding questions. ***1 minute explanation demonstration*** If coders figured out how to get their sprite to do something similar, have them document in their journal, share with a partner, or have a volunteer	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem ● **1A-AP-11** Decompose (break down) the steps needed to solve a problem into a precise sequence of instructions. ● **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops. **Practices reinforced:** ● Communicating about computing ● Testing and refining computational artifacts ● Creating computational artifacts **Concepts reinforced:** ● Algorithms ● Control **Video:** Suggestions for reverse engineering (4:25) **Note:** It is not recommended to show each of these ideas at once, but to show one idea, give time for application and exploration, show another idea, give time for application and exploration, etc. This process could take multiple classes. Also, some of these examples may be difficult for young coders, so go slow and encourage copying and modifying code as it's good practice. **Alternative suggestion:** If reverse engineering is too difficult for the coders you work with, you could display the source code and have coders predict what will happen.

show the class their code and thought processes that led to the code. Otherwise, reveal the code, walk through each step of the algorithm, and explain any new blocks.
 	
4+ minute application and exploration
Encourage coders to try something similar, and leave your code up on display while they work. Facilitate by walking around and asking questions about how coders might change their code so it's not the same as yours.
	Suggested guiding questions:
	● What kind of blocks do you think you might need to do something like that?
	● Do you see a pattern where we might use a repeat?
	● What trigger blocks do you think I used for that sprite?
	● Did I use one trigger block or more than one?
	    ○ What makes you think that?
	● Why does the boat have a stop block?
	    ○ It stops the repeat forever algorithm from running since the boat is now hidden. This is good coding practice because it saves processing power.
	
	Suggested application and exploration questions:
	● What other code blocks could you use?
	● What other sprites might use similar code?
	● What sounds might you hear in your interactive diorama?
	    ○ How will different sprites make different sounds?
	● What background might the diorama have?
	    ○ Do you need to create your own or can you use one already made?

<u>Adding even more (5+ minutes)</u>:
If time permits, encourage coders to explore what else they can create in ScratchJr. Although future lessons will explore different features and blocks, early experimentation should be encouraged.
 	
While facilitating this process, monitor to make sure coders don't stick with one feature for too long. In particular, coders like to edit their sprites/backgrounds by painting on them or taking photos. It may help to set a timer for creation processes outside of using blocks so coders focus their efforts on coding.
	Standards reinforced:
	● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem
	Practices reinforced:
	● Testing and refining computational artifacts
	● Creating computational artifacts
	Concepts reinforced:
	● Algorithms
	● Control
	
	Suggested questions:
	● What else can you do with ScratchJr?
	● What do you think the other blocks do?
	    a. Can you make your sprites do ____?
	● What other sprites can you add to your project?
	    a. What else might you want to do when you grow up?
	● What sounds might we hear?
	● What other trigger blocks might we use?

<u>Similar projects:</u> Have coders explore the sample projects built into ScratchJr (or projects from other coders), and ask them to find code similar to what they worked on today.	**Standards reinforced:** ● **1A-AP-10** Develop programs with sequences and simple loops, to express ideas or address a problem **Practices reinforced:** ● Testing and refining computational artifacts **Concepts reinforced:** ● Algorithms **Note:** Coders may need a gentle reminder we are looking at other projects to get ideas for our own project, not to simply play around. For example, "look for five minutes" or "look at no more than five other projects." **Generic questions:** ● How is this project similar (or different) to something you worked on today?

	• What blocks did they use that you didn't use? a. What do you think those blocks do? • What's something you like about their project that you could add to your project?

Differentiation

Less experienced coders	More experienced coders
ScratchJr is simple enough that it can be picked up relatively quickly by less experienced coders. However, for those who need additional assistance, pair them with another coder who feels comfortable working cooperatively on a project. Once coders appear to get the hang of using ScratchJr, they can begin to work independently.	Because ScratchJr is not inherently difficult, experienced coders might get bored with simple projects. To help prevent boredom, ask if they would like to be a "peer helper" and have them help out their peers when they have a question. If someone asks for your help, guide them to a peer helper in order to encourage collaborative learning. Another approach is to encourage experienced coders to experiment with their code or give them an individual challenge or quest to complete within a timeframe.

Debugging Exercises (1-5+ minutes each)

Debugging exercises	Resources and suggestions
Why doesn't the boat forever sail across the screen? • We need to use the repeat forever block instead of the end block (which is an optional block to indicate the end of an algorithm) Why does the shop get so big when pressed? • We need to set the second parameter to 1, not 10 Why doesn't the rocket fly into the sky and hide? • We need to change the move up parameter from 3 to 13 ScratchJr Debugging List	**Standards reinforced:** • **1A-AP-14** Debug (identify and fix) errors in an algorithm or program that includes sequences and simple loops **Practices reinforced:** • Testing and refining computational artifacts **Concepts reinforced:** • Algorithms • Control Display one of the debugging exercises and ask the class what they think we need to fix in our code to get our project to work correctly. Think out loud what might be wrong (e.g., did I use the wrong trigger block, did I forget to repeat something, did I put a block in the wrong place, am I missing blocks, etc.). Ask the class to talk with a neighbor how we might fix the code. Have a volunteer come up to try and debug the code (or demonstrate how). Repeat with each debugging exercise.

Unplugged Lessons and Resources

Standards reinforced:

- **1A-AP-08** Model daily processes by creating and following algorithms (sets of step-by-step instructions) to complete tasks

Although each project lesson includes suggestions for the amount of class time to spend on a project, BootUp encourages coding facilitators to supplement our project lessons with resources created by others. In particular, reinforcing a variety of standards, practices, and concepts through the use of unplugged lessons. Unplugged lessons are coding lessons that teach core computational concepts without computers or tablets. You could start a lesson with a short, unplugged lesson relevant to a project, or use unplugged lessons when coders appear to be struggling with a concept or practice.

List of 100+ unplugged lessons and resources

Reflection and Sharing	
Reflection suggestions	**Sharing suggestions**
Coders can either discuss some of the following prompts with a neighbor, in a small group, as a class, or respond in a physical or digital journal**.** If reflecting in smaller groups or individually, walk around and ask questions to encourage deeper responses and assess for understanding. Here is a sample of a digital journal designed for Scratch (source) and here is an example of a printable journal useful for younger coders. **Sample reflection questions or journal prompts:** ● How did you use computational thinking when creating your project? ● What's something we learned while working on this project today? ○ What are you proud of in your project? ○ How did you work through a bug or difficult challenge today? ● How did you help other coders with their projects? ○ What did you learn from other coders today? ● What's a fun algorithm you created today? ● What's something you could create next time? ● What questions do you have about coding? ○ What was challenging today? ● How might you use code when you grow up? ○ How could you use code for fun? ○ How could you use code in a future job? ● What did you want to do in your project that you couldn't figure out how to do with code? ● When would you want to trigger more than one algorithm at the same time? ● More sample prompts (may need adapting for younger coders)	**Standards reinforced:** ● **1A-AP-15** Using correct terminology, describe steps taken and choices made during the iterative process of program development **Practices reinforced:** ● Communicating about computing ● Fostering an inclusive culture **Concepts reinforced:** ● Algorithms ● Control ● Modularity ● Program development **Peer sharing and learning video:** Click here (1:33) At the end of class, coders can share with each other something they learned today. Encourage coders to ask questions about each other's code or share their journals with each other. When sharing code, encourage coders to discuss something they like about their code as well as a suggestion for something else they might add.